Journal of Information Processing Vol.23 No.1 58–66 (Jan. 2015)

[DOI: 10.2197/ipsjjip.23.58]

Regular Paper

Development of resource management server for production IaaS services based on OpenStack

Yoji Yamato[1,a)] Yukihisa Nishizawa[1] Masahito Muroi[1] Kentaro Tanaka[1]

Received: April 4, 2014, Accepted: September 12, 2014

Abstract: In this paper, we show the development of resource management server to enable production Cloud services easily based on OpenStack. In recent days, Cloud computing technologies have progressed and many providers have started Cloud services. Some providers use proprietary systems but others use open source IaaS software such as OpenStack and CloudStack. Because the community of OpenStack development is very active, we expect OpenStack will become a de facto standard open source IaaS software. Because OpenStack target is providing primitive APIs for IaaS control, there are some problems to use OpenStack as it is for production services. For example, there are some problems that logical/virtual resources CRUD transactions are insufficient, nova-scheduler which determines hypervisors for virtual machines deployment does not consider operators business requirements and logical checks of unsuitable API calls are insufficient. Therefore, we propose a resource management server which manages physical resources and logical/virtual resources to enable production IaaS services easily based on OpenStack. The resource management server mediates users and OpenStack, provides added actions such as logical checks of API calls, multiple API combination uses, scheduling logic of hypervisors for virtual machines. We implemented the proposed resource management server and showed that operators can operate reliable IaaS services without conscious of OpenStack problems. Furthermore, we measured the performance of multiple API combination uses and showed our method could reduce users waiting time of image deployment or image extraction from volume.

Keywords: OpenStack, Cloud computing, IaaS, resource management server

1. Introduction

In recent days, Cloud computing technologies such as virtualization and scale-out have been progressed and many providers have started Cloud services. According to the definition of the United States NIST [1], Cloud service models can be divided SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). IaaS provides hardware resources of CPU or Disk via a network. For examples, Amazon Web Services EC2 (Elastic Compute Cloud) [2] and Rackspace Public Cloud [3] are production IaaS services.

For setting up IaaS cloud software, Amazon uses proprietary systems and RackSpace uses open source software OpenStack [4]. OpenStack, CloudStack [5] and Eucalyptus [6] are major open source IaaS software and adoptions of open source IaaS software are increasing. Recently, OpenStack development community is very active and new functions are released every 6 months. We also expect the openness of OpenStack developments and develop IaaS services based on OpenStack.

However, OpenStack main target is providing primitive APIs of IaaS control. There are some problems when operators provide reliable Cloud IaaS services using only OpenStack. For examples, logical/virtual resources CRUD transactions are insufficient and unnecessary resources are remained when API terminates abnormally, nova-scheduler which determines hypervisors for VM (virtual machine)s selects hypervisors randomly and does not consider operators business requirements, logical checks of unsuitable API calls are insufficient such as volume deletion API can be done when a snapshot is creating from the volume.

For this reason, we propose a resource management server (RM) which manages physical resources and logical/virtual resources both to enable production IaaS services easily based on OpenStack. The RM mediates users and OpenStack, provides added actions such as logical checks of API calls, multiple API combination uses, scheduling logic of hypervisors for VMs. The RM makes users can use reliable IaaS services. We design and implement the proposed RM and show it can support the IaaS service operations without conscious of OpenStack problems. Furthermore, we measure the performance of multiple API combination uses and show our method can effectively reduce users waiting time of image deployment or image extraction from volume.

The rest of the paper is organized as follows. In Section 2, we introduce OpenStack architecture and clarify OpenStack problems for production business use. In Section 3, we propose a RM which mediates users and OpenStack, and also study policies to resolve OpenStack problems using RM. We explain hypervisors scheduling for VMs and high speed image deployment/image extraction from volume through multiple API combination use in detail. In Section 4, we implement the RM, confirm our proposed methods feasibility and evaluate the performance. We compare our work to related works in Section 5. We summarize the paper in Section 6.

[1] NTT Software Innovation Center, NTT Corporation, Musashino, Tokyo 180–8585, Japan

[a)] yamato.yoji@lab.ntt.co.jp

2. OpenStack Problems for Production IaaS Services

2.1 Outline of OpenStack

OpenStack, CloudStack and Eucalyptus are major open source IaaS software, and among them OpenStack community is active because many providers contribute developments and adopted services are rapidly increasing. **Figure 1** shows architecture of OpenStack.

OpenStack is composed of the function block which manages each logical/virtual resource, and the function block which provides Single Sign On authentication among other function blocks.

Neutron controls virtual networks. OVS (Open Virtual Switch) [7] and other software switches can be used as a virtual switch. Nova controls VMs. KVM (Kernel based Virtual Machine) [8], Xen [9] and others can be used as a hypervisor of a VM. OpneStack provides two storage management function blocks. Cinder for block storages and Swift for object storages and both storages are used for retaining data. Block storages can be attached to VMs like local disk volumes. Glance manages image files for VMs. An image file such as OS image can be deployed to a block storage using Cinder and Glance and a user can boot a VM of certain OS from the block storage using Nova. Keystone is a base which performs integrated authentications of these function blocks. The functions of OpenStack are used through REST (Representational State Transfer) APIs. There is also Web GUI called Horizon to use the functions of OpenStack.

OpenStack major version is released once a half-year and the latest version name is IceHouse. After IceHouse, the new functions to catch-up Amazon EC2 will be added. However, compared with Amazon EC2 or CloudStack, it can be said that OpenStack is still in developing phase. Therefore, we think there are some problems to adopt OpenStack as it is for production IaaS services.

2.2 Clarification of OpenStack Problems

As mentioned above, there remain insufficient points of OpenStack functions. In particular, since OpenStack main targets are providing primitive APIs to manage logical/virtual resources, there are some insufficient points for a reliable IaaS service. A reliable IaaS service has some features. It is running for a long period of time, it has a sufficient high performance/availability, a prescribed fee is charged per resource usage and physical resources are shared effectively to reduce a prescribed fee. Here we clarify 6 major problems when we use OpenStack for reliable IaaS services. 1 to 3 are from OpenStack insufficient functions. 4 to 6 are from operator business requirements. OpenStack improvement has been progressed in each new version, but these problems have not been completely resolved yet.

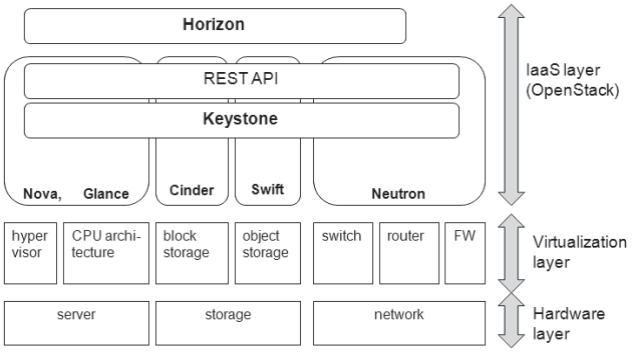


Fig. 1 OpenStack architecture.

2.2.1 Logical Checks of OpenStack API Calls are Insufficient

OpenStack provides general purpose APIs and have few precondition checks, there are problems that logically inappropriate API calls can be received. For example, a volume deletion API call can be performed during creating a snapshot from the volume. This causes a snapshot creation process incomplete. Since the logical checks of API calls are insufficient, there are problems in the view of user convenience because API processing may terminate abnormally after API request is received.

Insufficient logical checks of API calls cause failures of resource provisioning for orders and unnecessary resources remained. These increase risks of wrong charging and performance/availability degradation.

2.2.2 Transaction Management is Insufficient

Although OpenStack performs an asynchronous processing correspond to an API call, transaction management is insufficient. Therefore unnecessary resources remain in OpenStack side when OpenStack API processing terminates abnormally. For example, if a problem occurs when a volume creation API is performing, unnecessary volume may remain in OpenStack side.

Insufficient transaction managements cause unnecessary resources remained and unnecessary resourses increase risks of wrong charging and performance/availability degradation.

2.2.3 Completion Confirmation of Neutron Asynchronous Processing is Insufficient

As a design concept, Neutron manages a resource state in DB and does not care an actual completion time after it has written the requests to DB. The thread of processing a network resource checks DB periodically and if there are pending requests, it carries out the request asynchronously. For example, when the security group setting (in other words, access control setting) of a logical router is performing in Neutron, security group setting may not complete yet even if the API requester has received the response of DB written. This increases a risk of allowing malicious access.

Neutron does not guarantee when actual virtual network resources are created, operators cannot charge usage fees accurately.

2.2.4 OpenStack Does Not Manage Physical Resources

OpenStack manages virtual or logical resources and its management of physical resources is insufficient. Ironic [10] is a project of OpenStack for provisioning on physical machines but its implementation is undergoing. For IaaS services operation, operators need to manage physical resources states. Since it is a mission of operator to provide services stable for a long time, physical resources maintenances are important. For example, we need operations of migrating VMs to other physical servers temporally and blocking new VMs deployment on the physical server during the maintenance of physical server replacements.

Insufficient managing of physical servers and virtual resources may lead a delay of facilities extension and an inefficient physical resources sharing. These increase a risk of performance degradation.

2.2.5 VM Allocation Scheduler Does Not Consider Business Requirements of Operators

Intelligent VM allocation to physical servers is also insufficient in OpenStack. When nova-scheduler determines a hypervisor where a VM will be deployed, we can use Filter Scheduler [11]. Filter Scheduler can set 30 type conditions of hypervisor selections but it lacks some important conditions such as software license or VM usage conditions. We think it does not meet some business requirements to reduce license cost or noisy neighbor problem. For example, when guest OS of VMs are RedHat Enterprise Linux (RHEL), these VMs may need to be deployed on a hypervisor which host OS is RHEL. In other example, when there is a VM used much, operators would like to put the VM into a specific hypervisor to isolate.

If VMs are not deployed on physical servers effectively and physical resources are not shared effectively, IaaS service prices remain high because of facilities costs.

2.2.6 Naive API Calls May Have Problems of System Inconsistency or Lower Performance

Because OpenStack provides general and primitive APIs, naive API calls may bring problems that unnecessary resources are remained in OpenStack or user waiting time becomes too long. For example, when deleting a tenant (in other word, virtual floor), it is necessary to delete VMs or volumes under the tenant. However we can delete only a tenant by tenant control API, there may remain unnecessary resources in OpenStack. In other example, when we create a new volume from a certain OS image, the volume size is larger and larger, user waiting time becomes longer and longer. When volume size is 50 GB, it takes more than 15 minutes to create and it makes users frustrated.

Naive usages of OpenStack APIs take long times and decrease satisfactions of users.

Although there are other slight problems, we think main problems are these 6 items for reliable IaaS services on OpenStack. These 6 problems are common among other IaaS software (those do not have all of them but have many of them). CloudStack and Eucalyptus have same problems of operator business requirements 4-6. Regarding to OpenStack insufficient problems 1-3, CloudStack has 1 problem and Eucalyptus has 1 and 2 problems.

Of course, OpenStack insufficient functions are discussed in OpenStack community but the implementation is not enough yet. And some operator business requirements are out of scope in OpenStack community. Becuase we need to launch our IaaS services fast, we resolve 6 problems which cover insufficient functions and operator business requirements by ourselves. In parallel with our development, we also feedback these enhancements to OpenStack community. Taskflow is a project for transaction management in OpenStack and we are discussing its specification much in OpenStack community.

If we develop our codes inside OpenStack, there is a code gap between community OpenStack and our customized OpenStack. The gap may increase investigation efforts when we encounter an OpenStack problem because of complicated code structures. For this reason, we implemented these codes in a RM which is a server outside OpenStack.

3. Proposal of Resource Management Server

We propose a RM which manages physical resources and logical/virtual resources to enable production IaaS services easily based on OpenStack. The RM behaves as a proxy when users or operators use OpenStack APIs, analyzes requests and provides added actions such as logical checks. Because 6 problems depend on business types, we design the RM as an external function separated from OpenStack. Here, we describe the policies how to resolve 6 problems. Especially, we explain 5 and 6 policies in detail because these two problems need new methods.

3.1 Logical Checks of OpenStack API Calls

To resolve insufficient logical checks of OpenStack, the RM analyzes API requests and calls OpenStack APIs whether there is no problem of precondition or parameter validity.

Because all virtual/logical resource controls need logical checks, our RM checks more than 200 logics. Logical checks are divided into following patterns.

- Check elemental preconditions. (validities of Region, Availability Zone, Tenant, VM Flavor, etc)
- Check each resource is operable or not. (e.g., when we create a snapshot from a volume, the volume status needs to be valid)
- Check related resources before deletion. (e.g., during image extracting from a volume, the volume should not be deleted)
- Check parameter values validities. (e.g., VLAN ID, CIDR (Classless Inter-Domain Routing) IP format, etc)
- Check contract quota, system threshold of each resource.

Rigid logical checks of API calls can prevent unnecessary resources remaining and abnormal terminating after order acceptances.

3.2 Transaction Management of Virtual/logical Resources Operations

To resolve insufficient transaction management of OpenStack, the RM manages transactions and calls a reverse operation API if a transaction does not complete. OpenStack does not have purge APIs, we need to call a reverse operation API like deletion API when creation API error occurs. In this way, the RM prevents an inconsistency of virtual/logical states between OpenStack and API requester. Since OpenStack API is a small unit processing, the RM manages related API calls as a transaction and if one of processing fails, calls reverse operation APIs for all related APIs. For example, logical router setting needs plural APIs such as security group and routing settings.

Transaction managements can prevent unnecessary resources remaining.

3.3 Confirm Neutron Asynchronous Completion Notifications

Because an operator business is charging fees for provisioned resources, it is fatal to charge a resource not created yet. To resolve insufficient completion information of Neutron, the RM

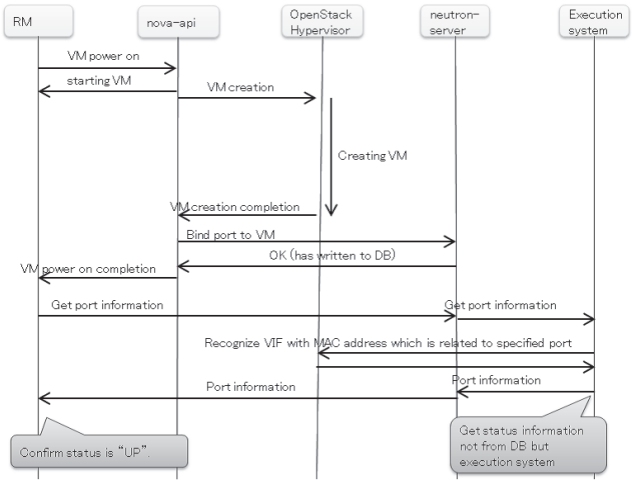


Fig. 2 Resource management server confirms VM port setting.

confirms actual resources validities, and then changes the resources statuses to valid in DB. All virtual/logical network resources are controlled by Neutron, about 40 operations are influenced. A confirmation method depends on each resource operation.

For example, we explain VM port setting confirmation using **Fig. 2**. When we call VM power on API, there is a possibility that VM is available but a port is not available and the VM cannot communicate other VMs. The RM calls an API to get the status of port related to the VM, confirms status is "UP," then changes the status of VM and notifies a completion to a user.

Confirming completions of Neutron virtual network resources certainly, operators can charge resource usage fees accurately.

3.4 Physical Resource Management

Because OpenStack physical resource management is insufficient, the RM manages physical resources such as physical storages, servers and retains mapping information what virtual/logical resources are on each physical resource. Specifically, the RM manages each physical server, each cluster of physical servers, mapping of OpenStack logical hosts and physical cluster, each physical storage, user volume or image area within physical storage. Adding physical resources information to OpenStack virtual/logical resources information, the RM can manage what virtual/logical resources are on each physical resource uniformly. In addition, the RM manages capacities that how much virtual resources deploy on each physical server and quota settings that how much virtual resources each user can create. This information of specifications, capacities or quota settings are set by operators.

Managing mapping information of physical servers and virtual resources, operators can use physical server effectively and extend facilities accurately.

3.5 Scheduling Hypervisors Where VMs Deploy

To resolve Nova Filter Scheduler insufficient points, the RM determines a hypervisor based on some consideration points where a VM deploys using physical server information. After determination, the RM requests a VM deployment to OpenStack

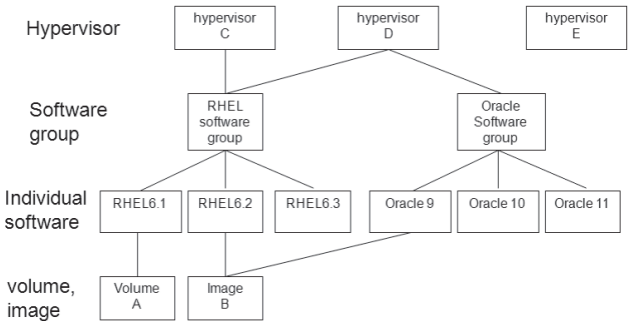


Fig. 3 Software license management relationships.

with concrete hypervisor assignment.

For operator businesses, there are four consideration points where to deploy a VM; license, VM usage, physical server capacity and migration availability.

- License.

A VM which uses license software may need to be deployed on specified hypervisors. For example, VM A uses RHEL6.1, VM B uses RHEL6.2 and VM C uses CentOS, we may need to deploy VM A and B on a hypervisor which host OS is RHEL.

- VM usage.

Operators need to consider some VMs are used frequently and consume resources much. Operators may need to isolate these VMs for minimizing bad effects to other VMs. Thus, the RM needs to deploy these VMs on specified hypervisors for isolation.

- Physical server capacity.

operators would like to reduce working physical servers and operation costs, the RM needs to take in VMs to one physical server as much as possible. And when there are plural servers which have margin rooms for VMs, the RM needs to select a hypervisor to enhance effectiveness of physical server resources usage. (Because some users stop VMs use, plural servers have margin rooms for new VMs)

- VM migration.

When a VM migrates to other hypervisors, we need to check a VM migration availability. KVM is a most used hypervisor in OpenStack community, but KVM specification prohibits VM migrations to other hypervisors with no CPU compatibility. Because OpenStack does not manage physical server CPU information, there is a risk of live migration or block migration failure after API calls.

Based on these consideration points, the RM manages hypervisor information of CPU type, resource capacity, usage; normal or isolation, state; under use or stock and what software group is related. This information is set by operators.

Figure 3 shows the concept of software license management. Multiple software licenses can be attached to bootable volumes or images. Software group is a bundle concept of individual software. And multiple software group can be related to a hypervisor.

In Fig. 3 case, RHEL6.1 is attached to volume A and RHEL6.2 and Oracle9 are attached to image B. RHEL6.1 is a component of RHEL software group. RHEL software group is related to hypervisor C and D. Then, the RM judges it can deploy the VM of volume A to hypervisor C or D. These relationships are managed in tables of resource management DB. Using software group con-

cept, we do not have to modify hypervisor settings but only modify software group setting when new software is released (e.g., RHEL 6.4).

Here, we explain a flow of hypervisor selection when normal VM activates. If there is another consideration point, we can put selection logic which corresponds to the point to these steps.

Step a - The RM narrows down hypervisors of "normal usage" from all hypervisors of specified Region and Availability Zone.

Step b - The RM narrows down hypervisors of "under use" state from step a candidates.

Step c - The RM narrows down hypervisors using software group information to satisfy license requirement from step b candidates.

Step d - The RM narrows down hypervisors using capacity and usage information from step c candidates. It means when the RM deploys a VM of specified flavor to the hypervisor, CPU core usage and Memory usage should not exceed the hypervisor thresholds.

Step e1 - The RM determines a hypervisor of Step d if there is only one candidate.

Step e2 - The RM determines a hypervisor based on specified calculation formula if there are plural candidates. This calculation formula can be customized by each provider. For example, we set the formula to bring $\sum$ [Memory size of VM]/ $\sum$ [CPU core number of VM] near to [Memory size of hypervisor]/[CPU core number of hypervisor] because CPU and Memory unbalanced usage does not use physical resources effectively. Here $\sum$ means summation of all VMs on the hypervisor.

Step f - When there is no candidate during b to e steps, the RM changes a stock hypervisor status from "stock" to "under use" and re-select a hypervisor from Step b.

In VM migration case, we add the step that the RM narrows down hypervisors using CPU type information whether there is CPU compatibility or not between current hypervisor and target hypervisor.

In isolation VM deploy or migration case, 1st step a is narrowing down hypervisors of "isolation usage." Because operators would like to isolate these VMs, step e2 calculation formula should be the logic that the RM distributes these VMs to candidate hypervisors as much as possible.

Deploying VMs considering software license and vacant capacity of physical servers, physical resources are shared effectively and operators can reduce prices of IaaS.

3.6 Multiple API Combination Use to Delete Unnecessary Resources or Enhance Performances

To resolve a problem that there remains unnecessary resources under a tenant when a tenant deletion API is called, the RM calls multiple deletion APIs collectively and clean up unnecessary resources. Specifically, the RM gets information of virtual/logical resources under a tenant when a tenant deletion API is requested, then calls each resource deletion API in accordance with 1) logical check rules. Virtual/logical resources are VMs, volumes, snapshots, logical switches, logical routers, floating IPs and others. After all virtual/logical resources are deleted, the RM deletes a tenant. When a user cancels a contract, the RM deletes all tenants in same way, deletes shared images among tenants, then deletes a user account.

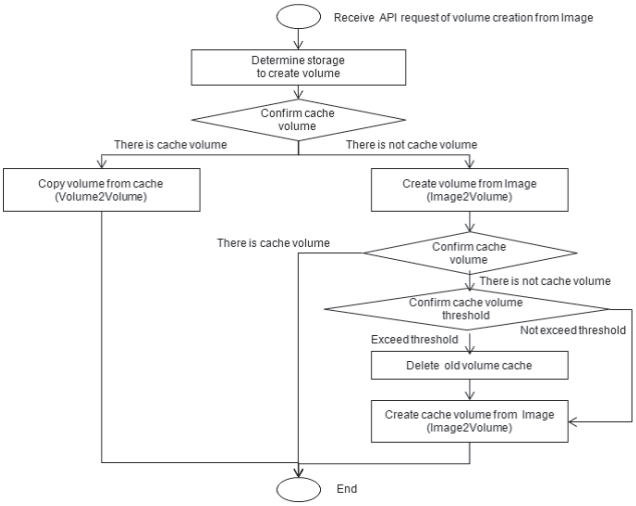


Fig. 4 Proposed image deployment flow.

To resolve a problem that large size image deployment and image extraction from volume take long waiting time for users, we propose a method to use multiple APIs appropriately for one image deployment or one image extraction from volume.

- The method to reduce image deployment time.

When we create a volume from an image using OpenStack Image2Volume API, 50 GB volume creation takes more than 30 minutes in Section 4 environment (Fig. 7). This is because it needs a cooperation of Glance and Cinder through a network.

Therefore, to reduce network traffic for frequently used images deployment case (e.g., create a new volume from OS image), we retain a created volume from a certain image as a cache volume in storages. And if a request of volume creation from the image comes, a volume is copied from the cache volume in the storage.

Figure 4 shows the flow of image deployment. If there is a cache volume, a new volume is created using Volume2Volume copy way from the cache volume. If there is not a cache volume, new volume is created using normal Image2Volume way and a cache volume is also created after user volume creation. Operators can set a threshold number of cache volume which is managed in the RM. And an old cache volume is deleted when a cache volume number exceeds the threshold value. Note that Volume2Volume copy is a physical storage function.

- The method to reduce VM power off time during image extraction from volume.

Users can recover VMs using an image of the attached volume. Periodical image extractions from volumes can be used for volume backup in case of volume failure. But when users extract an image from a volume, there is a risk of dirty volume and image extraction from volume failure in case of VM power on. Thus, operators recommend users to stop VM power before an image extraction from volume starts, and reactivate VM power on after the image extractions from volumes complete. However, an image extraction from a 50 GB volume takes 20-30 minutes in Section 4 environment (Fig. 7).

Therefore, to reduce VM power off time, we study the way not to extract an image from a volume directly but extract an im-

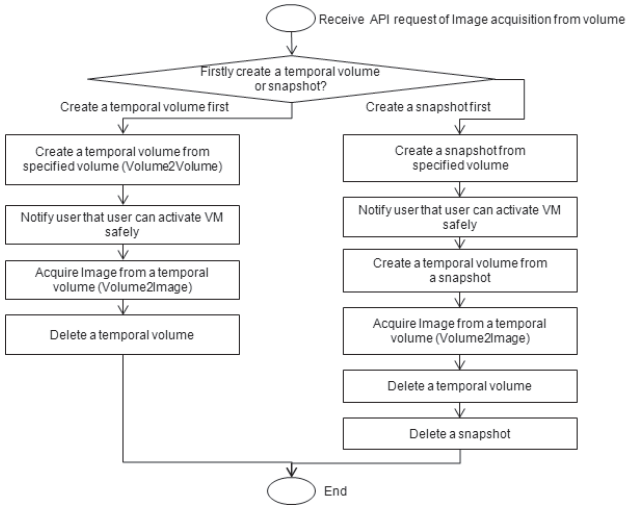


Fig. 5 Proposed image extraction from volume flow.

age from a temporal volume or snapshot. We create a temporal volume or snapshot from a volume first. These creation times are shorter than image extraction from volume, users can activate VM power on after these creations.

Figure 5 shows the flow of image extraction from volume. When the RM receives a request of image extraction from volume, the RM creates a temporal volume using Volume2Volume copy function or creates a snapshot using snapshot API. After this creation, the RM returns the information that users can activate VM safely. In a temporal volume case, the RM extracts an image from a temporal volume, deletes the temporal volume after the image extraction from volume complete. In a snapshot case, the RM creates a temporal volume from a snapshot, then extracts an image from a temporal volume, deletes the temporal volume and the snapshot after the image extraction from volume complete. Because a snapshot creation takes only a minute, users can activate a VM sooner but a total processing takes more time than Volume2Volume copy case. Thus, users can select these two types based on their needs (fast VM activate or fast total processing).

Combination uses of OpenStack API can reduce waiting times of image deployment or image extraction and enhance satisfactions of users.

4. Resource Management Server Evaluation

We implement the RM with proposed 1-6 functions and confirm it can support production IaaS service operations. Furthermore, we evaluate the performance of implemented RM.

4.1 Resource Management Server Implementation

Figure 6 shows the function block diagrams of the RM, OpenStack and related systems. The RM has three outer interfaces, Web GUI, API and a resident process of resource management. The RM manages virtual/logical resources and physical resources. It provides both GUI and API to users and operators for virtual/logical resources managements and provides GUI to operators for physical resources managements. Note that GUI for users can be customized by each provider using API functions. Requests from users or operators to OpenStack are put

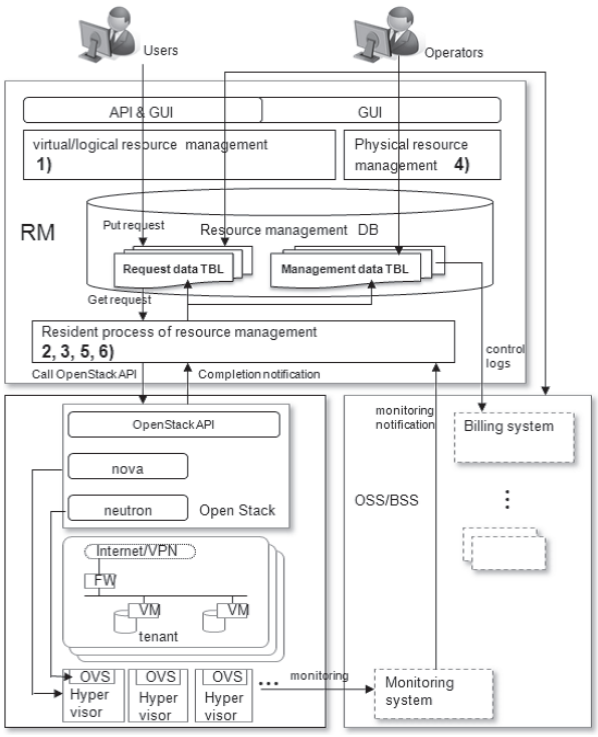


Fig. 6 Function blocks of resource management servers and related systems.

into requests data table in resource management DB first, then a resident process gets the requests and calls OpenStack APIs. The resident process of resource management receives event notifications from OpenStack such as VM creation completion and then put the state data to resource management DB. The resident process of resource management also receives event notifications of monitoring from OSS (Operation Support System) monitoring system which monitors physical servers, changes states of physical servers of DB. Resource control logs including Neutron asynchronous processing are put into resource management DB, operator BSS (Business Support System) billing system gets log data periodically for accountings. Proposed 6 functions are implemented on the function blocks described in Fig. 6.

We designed the RM on OpenStack and implemented it on Ubuntu 12.04 OS and Apache Tomcat 6.0.36 by Java language (JDK1.6.0.38).

We confirmed 6 functions validity on a test environment of performance measurement.

The transaction management function calls reverse APIs to roll-back virtual resources when OpenStack APIs processing fail. This function prevents unnecessary resources remained. Using the implemented RM, we have confirmed there is no unnecessary resource in OpenStack after roll-backs of failed OpenStack APIs processing. 30-40 asynchronous OpenStack APIs processing failures of Cinder, Glance, Keystone, Nova and Neutron were tested practically.

Regarding to rigid logical checks, we have confirmed that more than 200 logical checks block inappropriate API calls which may terminate abnormally in OpenStack beforehand by practical tests. For example, a request of VM live-migration is blocked during an image extraction of attached volume.

For the VM scheduling function tests, we created Windows VM and CentOS VM one by one. We have confirmed that Windows VMs are deployed on a physical server for Windows and CentOS VMs are deployed on a physical server for CentOS one by one and after "under use" physical servers are fulfilled, a "stock" physical server is activated and used for new VMs deployment. When a VM migrates to other physical servers, our RM schedules to migrate a VM to a physical server on which same software VMs are deployed. By practical tests, we have confirmed that same software VMs are deployed on a same physical server to reduce software license cost of physical server and VMs are deployed to a "stock" physical server after other "under use" physical servers are fulfilled to utilize physical servers effectively.

4.2 Performance Evaluation of Proposed Method

The RM reduces users waiting time of image deployment and image extraction from volume by multiple API combination uses. We measure the processing time and confirm our proposed method effectiveness.

4.2.1 Performance Measurement Patterns

Image deployment measurement.

- Volume creation time is measured when cache is hit and when cache is not hit.
- 10, 30 and 50 GB volumes are created from 10, 30 and 50 GB CentOS images.
- Concurrent processing numbers of volume creation are 1, 3 and 5.

Image extraction measurement.

- When we use a temporal volume, temporal volume creation time, image extraction from volume time and temporal volume deletion time are measured. When we do not use a temporal volume, image extraction from volume time is measured.
- Images are extracted from 10, 30 and 50 GB volumes.
- Concurrent processing numbers of image extraction from volume are 1, 3 and 5.

4.2.2 Performance Measurement Environment

Figure 7 shows performance measurement environments and **Fig. 8** shows each server specification and usage. Figure 7 omits maintenance servers such as syslog or backup servers, and redundant modules such as heartbeat. Two RMs compose Active-Active setting for redundancy. These servers are connected with Gigabit Ethernet.

4.2.3 Performance Measurement Results

Figure 9 (a) shows the performances of 50GB image deployment with different concurrent thread number and indicates that the caching mechanism can reduce processing times about 1/3 of no cache ones. When concurrent processing number is 1, volume creation time of cache use is 6.5 min and it is about 2/5 of no cache case. When concurrent number is 3, volume creation time of cache use is 8.6 min and it is about 1/3 of no cache case. We think OS images are frequently used for new volume creation timing. Therefore, operators prepare cache volumes from OS images beforehand, cache hit rate will become larger and processing time can be reduced sufficiently.

Figure 9 (b) shows the performances of 1 thread image deployment with different image size and indicates that the caching mechanism can reduce processing times about 1/3 of no cache ones. Of course, image sizes or concurrent thread number become larger, processing times also become larger. Figure 9 (a), (b) show that processing times are only 10 minutes when 50 GB image deployment with 5 concurrent threads and the performances are sufficient for practical use. These performance values are like Amazon Web Services image deployment values [12]. Thus, we evaluated our proposed cache method is effective to reduce processing times and shows good scalability for practical use.

Figure 10 shows the comparison of 50 GB image extraction processing times. When we use a temporal volume, total processing time becomes longer than direct image extraction time from the volume. However our target is to reduce a time of VM power off, users can activate a VM after a temporal volume creation. When concurrent processing number is 1, a temporal volume creation time is 6.6 min and it is about 1/3 of image extraction from

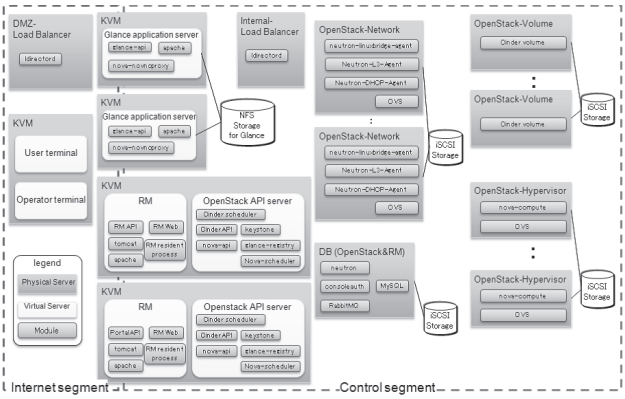


Fig. 7 Performance measurement environment.

Hardware	physical or VM	Name	Main usage	CPU model name	CPU core	RAM(GB)	HDD logical(GB)	NIC
HP ProLiant BL460c G6	physical	KVM host		Quad-Core Intel Xeon 2533 MHz×2	8	48	300	4
	VM	OpenStack API server	OpenStack stateless process such as API		assign: 4	assign: 8	assign: 60	
	VM	RM	proposed RM server		assign: 4	assign: 8	assign: 60	
HP ProLiant BL460c G6	physical	KVM host		Quad-Core Intel Xeon 2533 MHz×2	8	48	300	4
	VM	Glance application server	receive requests related to glance		assign: 8	assign: 32	assign: 150	
HP ProLiant BL460c G1	physical	DB (OpenStack&RM)	OpenStack and RM DB	Quad-Core Intel Xeon 1600 MHz×2	8	24	72	4
HP ProLiant BL460c G1	physical	OpenStack-Network	used for OpenStack logical network resources	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiant BL460c G1	physical	OpenStack-Volume	used for OpenStack logical volume resources	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiant BL460c G1	physical	OpenStack-Hypervisor	used for OpenStack VM resources	Quad-Core Intel Xeon 1600 MHz×2	8	24	72	4
IBM HS21	physical	DMZ-Load Balancer	Load Balancer for Internet access	Xeon E5160 3.0GHz ×1	2	2	72	1
IBM HS21	physical	Internal-Load Balancer	Load Balancer for Internal access	Xeon E5160 3.0GHz ×1	2	2	72	1
IBM HS21	physical	KVM host		Xeon E5160 3.0GHz ×1	2	2	72	1
	VM	User VM	VM for user terminal		assign: 1	assign: 1	assign: 20	
	VM	Operator VM	VM for operator terminal		assign: 1	assign: 1	assign: 20	
EMC VNX 5300	physical	iSCSI storage	iSCSI storage for user volume				500	
EMC VNX 5300	physical	NFS storage	NFS storage for Image				500	

Fig. 8 Each server specification and usage.

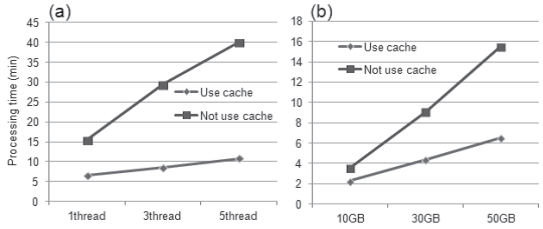


Fig. 9 Image deployment processing time comparison when we use a cache volume and do not use a cache volume. (a) 50 GB image deployment with 1, 3 and 5 concurrent thread number, (b) 1 thread image deployment with 10, 30 and 50 GB image size.

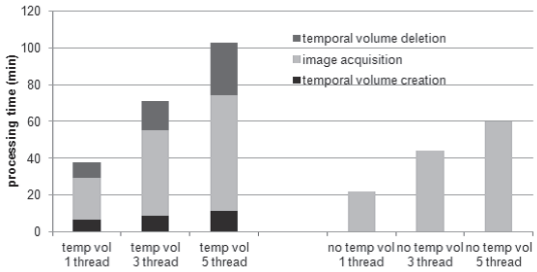


Fig. 10 50 GB image extraction each processing time comparison when we use a temporal volume and do not use a temporal volume.

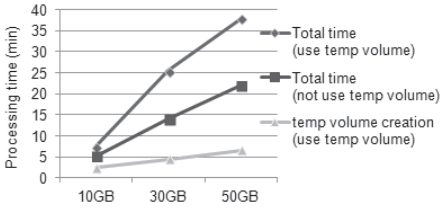


Fig. 11 Image extraction processing time with different volume size.

volume time from volume directly. When concurrent processing number is 3, a temporal volume creation time is 8.8 min and it is about 1/4 of image extraction time from volume directly.

Figure 11 shows the comparison of 1 thread image processing time with different image size. It indicates that total processing times of temporal volume use are about twice of no temporal volume use but VM power off times of temporal volume use are about 1/3 of no temporal volume use. Thus, we evaluate our proposed temporal volume method is effective to reduce a time of VM power off.

5. Related Works

Like OpenStack, OpenNebula [13], Eucalyptus [6] and CloudStack [5] are open source Cloud software. OpenNebula is a virtual infrastructure manager of IaaS building. OpenNebula manages VM, storage, network of company and virtualizes system resources to provide Cloud services. Eucalyptus characteristic is an interoperability of Amazon EC2, and Xen, KVM or many hypervisors can be used on Eucalyptus. Our group also contributes to developments of OpenStack itself. Some bug fixes and enhancements of OpenStack are our group contributions.

There are some researches of resource allocations on shared or VPS hosting to use physical server resources effectively [14], [15]. Our work rearranges VMs on OpenStack. And a validity check of VM migration is a different requirement from shared hosting and new rearrangement logic is needed. The papers of [16] and [17] are VM allocation technologies. The paper [16] is a research of dynamic resource allocations on OpenStack. The paper [17] is a research of VM consolidation on OpenStack and it consolidates VMs while SLAs of VM are kept. Our VM scheduling considers not only vacant capacity of physical server or migration availability but also deploying same software VMs to a same physical server as possible and activating a “stock” server after fulfilling other “under use” servers. These logics are novel ones which other works do not have so that our method can reduce software license cost of physical server and can use physical server resources effectively. Filter Scheduler [11] can describe various logics but cannot control VMs considering software license.

Next, we compare performance enhancements by API combination use with other technologies. To extract an image from volume certainly, other IaaS software also need to stop VMs. Our method is novel because it extracts an image from a temporal volume, reduces VM stop time and improves satisfactions of user. Regarding to the image deployment, VMware vSphere [18] also deploys from a cache. But VMware has some restrictions such as setting hypervisors to VMware ESX, our method does not have such restrictions because it only uses primitive and common IaaS APIs. The works of [12] and [19] show performances of plural IaaS platform. Our method showed sufficient performances of image deployment comparing with Refs. [12] and [19]. Naive API use of OpenStack may take long time for image deployment or extraction because OpenStack needs Glance and Cinder cooperation. Our work can reduce users waiting time by multiple API combination use while keeping interfaces for users.

Regarding to a total function of mediating IaaS and users, Amazon OpsWorks [20] is a managing function of IaaS. Amazon OpsWorks supports automatic scheduling or deploying on Amazon Web Services. Our RM provides reliable IaaS services by covering insufficient points of IaaS software and operator business requirements, and the design of RM can re-use for other Cloud software CloudStack or Eucalyptus because the design is based on primitive and common APIs of IaaS software.

Cloud API had discussed in DMTF (Distributed Management Task Force) CMWG (Cloud Management Working Group) and OGF (Open Grid Forum) OCCI (Open Cloud Computing Interface) WG. OCCI summarized the use cases of Cloud computing, required conditions of IaaS interfaces and evaluated each production Cloud management API whether it satisfies the required conditions or not. Elastic Hosts, Sun, IBM and other companies discussed in OCCI and the specification was submitted to CMWG. VMware, HP, Telefonica, Red Hat, Fujitsu, and Oracle have also proposed Cloud management API to CMWG (Delta Cloud, etc) and API standardization is expected. libcloud [21] and Simple Cloud API [22] are open source Cloud management APIs. libcloud is an open source library by Cloudkick for using two or more cloud services such as Amazon EC2 and Rackspace based on common APIs. Simple Cloud API are abstract APIs of storages, document database, queue services which Amazon EC2, Windows Azure, Rackspace Cloud Files and others provide.

6. Conclusion

In this paper, we proposed the RM which mediated users and OpenStack to enable production IaaS services based on OpenStack. To resolve problems of using OpenStack for reliable businesses, we designed the RM which managed both physical resources and logical/virtual resources and which also provided added actions such as logical checks of API calls, multiple API combination uses, scheduling logic of hypervisors for VMs. We implemented the proposed RM on OpenStack, confirmed functions feasibilities and measured performances.

It was confirmed that the RM enabled operators could operate without regard to 6 problems of OpenStack. We also con-

firmed our proposed scheduler could determine hypervisors for VMs meeting with requirements of license, VM isolation and VM migration. Moreover, we showed the effective reduction of users waiting time by multiple API combination uses. About 2/3 of image deployment waiting time was reduced by using a cache volume. Also 2/3-3/4 of image extraction from volume waiting time was reduced by using a temporal volume.

NTT group has started IaaS services based on OpenStack in 2013, and our RM has been used in the services. In the future, we will modify the RM for OpenStack new versions. IceHouse or Juno are new major versions of OpenStack and provides new functions to catch up Amazon EC2 such as Amazon CloudFormation and Simple Queue Service. We will also plan to add new additional functions of RM, for example a function of image sharing among users. In parallel with new functions implementing, we will hear comments of actual users who use OpenStack IaaS services and enhance the software quality or functionalities of RM.

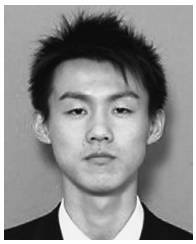
Acknowledgments We are thankful to Hiroshi Sakai and Hikaru Suzuki who are managers of this development.

References

[1] Mell, P. and Grance, T.: *The NIST Definition of Cloud Computing v15*, National Institute of Standards and Technology (Oct. 2009).
[2] Amazon Elastic Compute Cloud web site, available from ⟨http://aws.amazon.com/ec2⟩.
[3] Rackspace public cloud powered by OpenStack, available from ⟨http://www.rackspace.com/cloud/⟩.
[4] OpenStack, available from ⟨http://www.openstack.org/⟩.
[5] CloudStack, available from ⟨http://CloudStack.apache.org/⟩.
[6] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L. and Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing System, *Proc. Cloud Computing and Its Applications* (Oct. 2008).
[7] Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M. and Shenker, S.: Extending Networking into the Virtualization Layer, *Proc. 8th ACM Workshop on Hot Topics inNetworks* (*HotNets-VIII*) (Oct. 2009).
[8] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: Kvm: The Linux virtual machine monitor, *OLS '07: The 2007 Ottawa Linux Symposium*, pp.225–230 (July 2007).
[9] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles* (*SOSP'03*), pp.164–177 (Oct. 2003).
[10] Ironic, available from ⟨https://wiki.openstack.org/wiki/Ironic⟩.
[11] Filter Scheduler, available from ⟨http://docs.openstack.org/developer/nova/devref/filter_scheduler.html⟩.
[12] CloudHarmony cloud benchmark, available from ⟨http://cloudharmony.com/benchmarks⟩.
[13] Milojicic, D., Llorente, I.M. and Montero, R.S.: OpenNebula: A Cloud Management Tool, *IEEE Internet Computing*, Vol.15, No.2, pp.11–14 (Mar. 2011).
[14] Urgaonkar, B., Shenoy, P. and Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms, *Symp on Operating Systems Design and Implementation*, pp.239–254. ACM Press (2002).
[15] Liu, X., Zhu, X., Padala, P., Wang, Z. and Singhal, S.: Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform, *Proc. IEEE Conference on Decision and Control*, pp.3792–3799 (2007).
[16] Wuhib, F., Stadler, R. and Lindgren, H.: Dynamic resource allocation with management objectives - Implementation for an OpenStack cloud, *Proc. Network and Service Management, 2012 8th International Conference and 2012 Workshop on Systems Virtualiztion Management*, pp.309–315 (Oct. 2012).
[17] Corradi, A., Fanelli, M. and Foschini, L.: VM consolidation: A real case based on OpenStack Cloud, *Elsevier Future Generation Computer Systems*, DOI: http://dx.doi.org/10.1016/j.future.2012.05. 012, 2012 (June 2012).
[18] VMware vSphere, available from ⟨http://www.vmware.com/products/vsphere/⟩.
[19] Steinmetz, D., Perrault, B.W., Nordeen, R., Wilson, J. and Wang, X.: Cloud computing performance benchmarking and virtual machine launch time, *Proc. 13th Annual Conference on Information Technology Education* (*SIGITE '12*), pp.89–90 (2012).
[20] Amazon OpsWorks, available from ⟨https://aws.amazon.com/opsworks/⟩.
[21] libcloud, available from ⟨http://incubator.apache.org/libcloud/index.html⟩.
[22] Simple Cloud API, available from ⟨http://www.simplecloud.org/api⟩.

Yoji Yamato received his B.S., M.S. degrees in physics and Ph.D. degree in general systems studies from The University of Tokyo, Japan in 2000, 2002 and 2009, respectively. He joined NTT Corporation, Japan in 2002. There, he has been engaged in developmental research of Cloud computing platform, Peer-to-Peer computing, Service Delivery Platform and Semantic Web Services. Currently he is a researcher of NTT Software Innovation Center. Dr. Yamato is a member of IEEE and IEICE.

Yukihisa Nishizawa joined NTT Corporation, Japan in 2002. There, he has been engaged in developmental research of Cloud computing platform. Currently he is a researcher of NTT Software Innovation Center.



Masahito Muroi joined NTT Corporation, Japan in 2011. There, he has been engaged in developmental research of Cloud computing platform. Currently he is a researcher of NTT Software Innovation Center.

Kentaro Tanaka joined NTT Corporation, Japan in 2012. There, he has been engaged in developmental research of Cloud computing platform. Currently he is a researcher of NTT Software Innovation Center.